commodore

comments and bulletins concerning your COMMODORE PET™

The Transactor

BULLETIN # 9

PET™ is a registered Trademark of Commodore Inc.

February 28, 1979

Enclosed in this month's "Transactor" is a copy of Jim Butterfield's memory map. Generally it's a listing of what is contained in PET memory and where. Although other maps have been published in previous bulletins, I found Jim's to be the most comprehensive thus far. It lists nearly all of the sub-routines that PET has in ROM and also the areas of RAM that PET uses as registers and buffers. For those who haven't used a memory map it's as easy as using a city road map. We'll explore this simplicity with a few examples but first a brief explanation of PET memory and the memory map.

ADDRESSING

Every memory location in your PET contains one byte of information. In order for PET to get at these bytes it must have a means of accessing them. Therefore each and every memory location has its own individual address; all 65536 of them. The microprocessor places these addresses on the address buss which immediately enables one memory location to the data buss. Bearing that in mind, one of two operations can happen now. PET can either place a byte into that location (i.e. POKE) or "look" at what's already there (i.e. PEEK). When performing the first operation the microprocessor places a byte on the data buss and transfers it along the buss and into the enabled memory location.

In the second operation, the information or byte in the enabled location is transferred onto the data buss and along

...2/

the data buss back to the microprocessor. This location is not "emptied" but rather only a duplicate or copy of the information is transferred. Once either of these operations is complete the microprocessor then places a new address on the address buss and another location is enabled. This process repeats thousands of times every second, however these operations aren't possible on all memory locations, but I'll explain this later.

The microprocessor has control of 99.9% of the addresses being placed on the address buss. That extra 0.1% control was left for the user and can be obtained through use of the PEEK, POKE and SYS commands. When executing these commands the user must choose an address. This address will be one of the 65,536 memory locations (i.e. 0 to 65535). This is where the memory map enters the picture. The memory map may well be your most powerful tool for choosing addresses. If you look at the map you'll see that all of the addresses are listed in ascending order down the left hand side; first in hexadecimal and then in decimal. (See section on hexadecimal and binary for explanation of this conversion) the decimal address is the one you use when executing the above 3 BASIC commands. To the right are the descriptions of what you can expect to find at the corresponding addresses. If we then PEEK these addresses we are returned the actual bytes that are in those particular memory locations. For example, let's say during a program we hit the STOP key and got:

```
BREAK IN 600
READY.
```

PET gets '600' from a storage register at addresses 138 and 139. We could also PEEK these locations and find that 600 is indeed stored in 138, 139. However it is not stored as a six, a zero and a zero. Instead it is stored as the decimal conversion of

the line numbers representation in hexadecimal. All information of this type is returned in this manner. Now that we know what the memory map will help us do let's cover some of the rules.

RAM and ROM

We all go through life with basically 3 types of memory:

1. MEMORY PRESENT: This memory we use to remember things like what street we're driving on or our present location.
2. MEMORY PERMANENT: Things like our names and fire is hot we never forget.
3. MEMORY PAST: Recent occurrences and not so recent such as things we did 10 or 12 years ago.

In the PET there are only two:

1. RAM Random Access Memory: This type of storage is used for our programs and things that change such as the clock and previous line number.
2. ROM Read Only Memory: This is PET's permanent memory. In ROM are the addition routines, clock updating routines and loading routines to name a few. These functions would have to programmed into PET on each power up if they weren't permanently 'burnt in'.

The third type, memory past, is instantly 'forgotten' on power down. The only way to recall it is to first save it on tape, disc, etc.

Recall earlier I mentioned that POKE and PEEK aren't possible on all memory locations for several reasons:

A. Not all PET memory locations actually exist. On the memory map, locations 1024 to 32767 is the 'available RAM including expansion'. If you have a PET with 8K, simple arithmetic

4.../

shows that 3/4 of the available RAM space is non-existent. If you decide to expand your system, PET will 'fit' the added RAM into this area. However POKing or PEEKing this space (i.e. 8192 to 32767) will return invalid results on 8K PETs.

B. The same concept applies to locations 36864 to 49151. This is the available ROM expansion area.

C. Next on the memory map is the Microsoft BASIC area; locations 49152 to 57463. This is the memory that recognizes and performs your commands. Changing the contents of these locations is impossible because it is Read Only Memory and is actually 'burnt in' at the factory. Therefore, POKing these locations will simply do nothing. Also, Microsoft requested that these locations return zeros if PEEKed (for copyright reasons).

With these 3 rules and your memory map you are now equipped to explore capabilities of your PET that you probably never thought possible. Before we try some examples let's go into one more important occurrence that may have had you scratching your head ever since that first power up.

MISSING MEMORY?

When you turn on your 8K (where **K** = 1024) PET, the first thing it tells you is 7167 BYTES FREE; a reduction of almost 12%.

Q. Where did the missing 1024 bytes go?

A. It's still there...right below the available RAM space (notice it starts at location 1024). PET uses this memory to do some very useful operations for you which you can find and access by looking them up on the memory map.

Q. But why not do this in ROM space?

A. PET needs RAM type memory to store this data because it is always changing. The information in this "low" end of memory is actually produced by routines found in ROM.

5.../

Take for example the built-in clock. The clock or time is stored in locations 512, 513 and 514 of RAM. However the data comes from a routine found in ROM at location $F736_{hex}$. The time is of course always changing, therefore it must be stored in RAM. But because it is in RAM, you may also change it; either by setting TI or TI$ or you can POKE the above 3 locations. Try it.

Now let's try some examples.

1. Location 226 (00E2 in HEX) holds the position of the cursor on the line. Try these:

   ```
   POKE 226,20:?"PRINTS AT NEXT SPACE
   ?"123456789";:?PEEK(226)
   ```

2. Location 245 (00F5 in HEX) stores the line the cursor is presently on (0 to 24). POKing this location will move the cursor to the specified line after a display execution. For example try:

   ```
   ?"A": POKE 245,10:?"B":?"C"

   POKE 245,21-1:?"cu":POKE 226,20:?"PRINTS HERE"
   ```

   The above will move the cursor to line 20 (21-1), print a 'cursor up' on line 21 and display your message starting at column 21, line 20.

   While experimenting with out-of-range values I obtained some rather interesting results. Try POKing location 245 with a number greater than 24, say 40 or 60, and hit the cursor up/down key a number of times. Also, experiment with unusual numbers in location 226 such as:

   ```
   POKE 226,100:?"123456789"
   ```

6.../

3. Location 526 is the reverse field flag. POKing this address with a non-zero value will execute the following same line print statements in RVS field. Once finished, PET resets 526 to zero. Try this:

   POKE 526, 1:?"123":?"456"

   now INST a semi-colon between 3" and the colon (i.e. ...23";:?"4...) and re-execute.

4. Notice below the RVS field flag is location 525; the number of characters in the keyboard buffer. Above the RVS flag is the buffer itself at locations 527 through 536. Although this designates 10 buffer locations, there are actually only 9. The tenth (536) is for some reason a "dead" location. During program execution, the operating system scans the keyboard every 60th of a second. If keys are typed say during a 'FOR-NEXT' loop, they are stored in the keyboard buffer until the program encounters a GET or an INPUT.* PET then 'draws out' the contents of the buffer and implements them according to the command involved (GET or INPUT). However, if more than 9 keys are typed during the loop, PET erases the entire contents of the buffer and continues to fill the buffer with the 10th character as if it were the first, and so on ("modulo 10").

   *or after a BREAK, READY.

...7/

In the command mode (i.e. when you're operating PET directly all typed keys go first into the keyboard buffer and then into screen memory or VIDEO RAM. However you may also load the buffer under program control by POKing the ASCII representations of the characters into sequential locations of the buffer. You must also increment by 1 the contents of 525 each time another character is POKed in, but remember -- not past 9. Page 6 of "Transactor" #2 contains a table of all the values for characters and commands. "Transactor" #1, page 12 lists some extras such as cursor controls and the RETURN key (13). Try the following endless loop. 145 is a cursor up

```
POKE 525,4:POKE527,145:POKE528,145:POKE529,145:POKE530,13
```

Some other interesting items are:

```
POKE59409,52 - Blanks screen
POKE59409,61 - Screen back on
POKE59411,53 - Turns cassette motor on
POKE59411,61 - Turns motor off
POKE59468,14 - Lower case mode
POKE59468,12 - Graphics mode
POKE537,136 -  Disables STOP key and clock
```

If anyone knows of or discovers any peculiarities by "POKing" around, please send them in. When I receive enough of them a handy dandy 'PETRIX' card will be included in a future "Transactor" bulletin.

THE SYStem COMMAND

On the last three pages of the memory map are listings of the subroutines stored in PET ROM that perform your commands and programs. These subroutines are stored as machine language.

...8/

When a SYS command is executed PET jumps to the specified decimal address and continues from there in machine language. Take for example the Machine Language Monitor program. This is a machine language program and is initialized by a SYS command stored as a BASIC program line. LOAD and RUN your M.L.M. then type 'X' and hit 'RETURN' to exit to BASIC. Now list. What you'll see is:

```
10 SYS (1039)
```

Location 1039 is the address to which PET will jump and also the address at which the first maching language instruction is stored. (A listing of all of the M.L.M. instructions is in "Transactor" #5, pages 5A and 5B) . When this BASIC line is executed PET operates in machine code beginning with address 1039.

The SYS command does not require brackets around the specified address.

Since PET has its subroutines stored in machine language you can use the SYS command to access and execute them. However you may come up with some rather peculiar if not disastrous results. When jumping into ROM you may find yourself in the middle of a subroutine or at the beginning of a subroutine belonging to a major function routine. Often PET will 'hang-up' or crash and you will be forced to power down to resume normal operation. To demonstrate jumping into the middle of a routine, try the following examples:

1. SYS52764 (CE1C)
2. SYS62498 (F422)
3. POKE523,1:SYS62498 (F422)
4. SYS62463 (F3FF)
5. SYS64824 (FD48)

...9/

The numbers on the right are the addresses of the above sub-routines in hexadecimal. Compare them to the memory map, especially for e.g. #1. Also take a look at 523.

The following are examples of valid locations which you can use with the SYS command to access useful routines, however these routines are already accessible through BASIC.

1. SYS62651 (F346)
2. SYS62278 (F4BB)
3. SYS63134 (F69E)

Example #3 will perform a 'SAVE' but will not produce a tape header.

Experiment with your memory map. Hex to decimal conversions can be obtained using the method following this article.

SUMMARY

This has been merely 'a scratch on the surface' of the extremely complex inner workings of PET. Do not be afraid to experiment with the POKE and SYS commands. There is absolutely nothing you can do to harm PET from the keyboard that turning power off and on won't fix. Also do some PEEKing around especially in low end memory. One good way is to write a small monitor program:

```
10?"c"PEEK(516):GOTO 10
```

The above will monitor the 'SHIFT' key. Try running it and depress 'SHIFT'. Compare the map.

...10/

When POKing or SYSing to random addresses, remember the address you choose. Often PET will do something which may erase the address from the screen (e.g. SYS64840).

The addresses that have been listed here are only a few of many that are already known and only a minute percentage of the ones not known. Probe around and send in any discoveries, useful, peculiar or otherwise. They will be collected together and published in a future "Transactor" bulletin.

Karl J.

BINARY to HEXADECIMAL to DECIMAL

We all know how to count in base 10 or decimal. We start at zero and count one...two...three and so on to nine. Once nine is reached we've run out of numbers, that is single digit numbers. So in order to continue we must now make use of two digits; we place a "1" in the 10's column and reset the 1's column back to zero. Continuing from here, sooner or later we would reach 99. Adding "1" would generate a carry into the 10's column and this in turn will generate a carry into the 100's columns to zeros.

This explanation of base 10 was given simply to demonstrate how we actually do our counting that we just do naturally. Binary is much simpler than decimal because there are only two numbers to worry about; zero (0) and one (1).

Base 2 number set	Base 10 number set
0, 1	0, 1, 2, 3, 4, 5, 6, 7, 8, 9

With a little practise you'll see that counting in Binary is just as easy as counting in decimal.

Binary is base 2 ('Bi') just as decimal is base 10 ('Deci') just as hexadecimal is base 16 ('Hexadeci') but I'll talk about the "HEX" numbering system later.

In base 10 we are 'allowed' to count up to 9 before carrying the "1" into the next column. Generally in any base we count to one less than the base # and generate a carry into the next column. In base 2 we count up to "1" and do our carry. Just as we cannot fit a "10" base ten into one column we cannot fit a "2" base two into one column. The base # is most important. Let's illustrate by comparison.

....12/

NUMBER	NUMBER REPRESENTATION IN:	
	BASE 2	BASE 10
0	0000	0000
1	0001	0001
2	0010	0002
3	0011	0003
4	0100	0004
5	0101	0005
6	0110	0006
7	0111	0007
8	1000	0008
9	1001	0009
10	1010	0010
11	1011	0011

Notice how in binary, on every multiple of 2 a carry is generated whereas in decimal the carry is generated upon multiples of 10.

Let's now define the columns of the two number bases. In base 10 we have the 1's column, 10's column, 100's column and so on. Each column is the previous column times ten; 1 = 0.1 x 10, 10 = 1 x 10, 100 = 10 x 10 and so on. We can also represent these using exponents; $1 = 10^0$, $10 = 10^1$, $100 = 10^2$ (10 'squared'), $1000 = 10^3$ (10 'cubed'), and so on. In base two each column is the previous column times two; we have the 1's column, 2's column, 4's column, 8's, 16's, 32's and on. Using exponent representation, $1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, $32 = 2^5$, and so on. Now let's represent some numbers of the two bases using their column breakdown:

...13/

$2_{\text{base }10} = 0002 = 0*1000 + 0*100 + 0*10 + 2*1$

$2_{\text{base }2} = 0010 = 0*8 + 0*4 + 1*2 + 0*1$

$7_{10} = 0007 = 0*1000 + 0*100 + 0*10 + 7*1$

$7_2 = 0111 = 0*8 + 1*4 + 1*2 + 1*1$

$12_{10} = 0012 = 0*1000 + 0*100 + 1*10 + 2*1$

$12_2 = 1100 = 1*8 + 1*4 + 0*2 + 0*1$

The same three examples using exponent representation will be:

$2 = 0002 = 0*10^3 + 0*10^2 + 0*10^1 + 2*10^0$

$2 = 0010 = 0*2^3 + 0*2^2 + 1*2^1 + 0*2^0$

$7 = 0007 = 0*10^3 + 0*10^2 + 0*10^1 + 7*10^0$

$7 = 0111 = 0*2^3 + 1*2^2 + 1*2^1 + 1*2^0$

$12 = 0012 = 0*10^3 + 0*10^2 + 1*10^1 + 2*10^0$

$12 = 1100 = 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0$

$2^0 = 1$	$2^4 = 16$
$2^1 = 2$	$2^5 = 32$
$2^2 = 4$	$2^6 = 64$
$2^3 = 8$	$2^7 = 128$

Use this table as a reference for the following exercises.

Try the following example on representing decimal numbers in binary by placing a 1 or a 0 in the correct column position.

NUMBER		2^3	2^2	2^1	2^0
4	=	_	_	_	_
12	=	_	_	_	_
13	=	_	_	_	_
14	=	_	_	_	_
15	=	_	_	_	_

What must be done to represent the number 16 in binary. If you said " A fifth digit must be used at the leftmost position ", then you're absolutely right. Except for one thing: digit is a word we use in decimal. In binary we use the word BIT derived from Binary digIT. By implementing a fifth bit it is now possible to represent numbers greater than 16 but only up to 31. Once past 31, a sixth bit position must be used. Continue with the exercise. Notice the leftmost column values have changed.

NUMBER		2^5	2^4	2^3	2^2	2^1	2^0
16	=	_	_	_	_	_	_
21	=	_	_	_	_	_	_
28	=	_	_	_	_	_	_
32	=	_	_	_	_	_	_
51	=	_	_	_	_	_	_
62	=	_	_	_	_	_	_
63	=	_	_	_	_	_	_

...15/

What would be the highest posible number you could represent using only 6 bits?

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
? =			1	1	1	1	1	1

7 bits?

? =		1	1	1	1	1	1	1

8bits?

? =	1	1	1	1	1	1	1	1

If your answers were 63, 127 and 255, you're correct. Notice how these values are 1 less than the value of the next bit position to the left. (2^8 =256)

The BYTE

Every memory location in PET is actually one byte. A byte consists of 8 bits. In computer electronics the binary number system is used. This way we can use a high voltage to represent a "1" and a low voltage to represent a "0". Can you imagine the circuitry that would be required to operate a computer in decimal or base 10? Ten unique voltages would have to be used to represent each of the ten digits. Then a separate computer would probably be required to distingusih between them all. By using binary PET must only distinguish between two voltages. Since a 5 volt supply is used for the logic circuitry, anything over 2.4 volts is considered high or a "1" and anything under is considered low or a "0". These voltages are typically 4.8 volts and 0.2 volts, respectively. Each bit of every byte in memory holds one of these voltages. With 8 bits in each byte, 256 combinations can be obtained (0-255) as you can see from the above exercise. If you look at the table on page 6, "Transactor #2, you'll see that all the keys can be encoded into one of these combinations. PET uses some combinations to represent the commands so that they only take up one byte in memory. PET also uses some of these combinations twice to represent graphics as you'll see by comparing the table to page 12 of "Transactor" #1. PET ROM routines distinguish between commands and graphics.

...16/

Try POKing a RAM location, say 6000, with a number greater than 255, say 256. A ?ILLEGAL QUANTITY ERROR will be returned because more than 8 bits are required to represent 256 in binary.

	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
256 =	1	0	0	0	0	0	0	0	0

Essentially, 256 won't 'fit' into a single byte.

Try PEEKing a non-existent memory location, say 10,000:

```
?PEEK(10000)
```

A 255 will be returned. A unconnected or open line is considered high by PET. Since the byte is not really there, the data buss lines will be open and read as high or all 1's by the micro-processor.

Hexadecimal or "HEX"

Hexadecimal means base 16. This means we can count up to 15 before generating a carry. However we can't use the numbers 10, 11, 12, 13, 14 and 15; these take up two columns. We need to represent these numbers using a single character. Therefore we use the first 6 letters of the alphabet.

Hexadecimal number set

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

$A_{16} = 10_{10}$	$D_{16} = 13_{10}$
$B_{16} = 11_{10}$	$E_{16} = 14_{10}$
$C_{16} = 12_{10}$	$F_{16} = 15_{10}$

...17/

When counting in HEX we generate our carry upon 16:

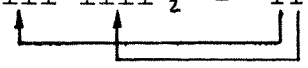
$$\begin{array}{rcl} 14_{10} & = & E_{16} \\ \downarrow & & \downarrow \\ 15_{10} & = & F_{16} \\ \downarrow & & \downarrow \\ 16_{10} & = & 10_{16} \end{array}$$

Recall in binary, 4 bits will yield a maximum of 15

$$15_{10} = 1111_2 = F_{16}$$

Now since a byte has 8 bits, we can split it up into two fields of our and then represent it as two hexadecimal characters.

$$\begin{array}{rcl} 4_{10} & = & 0000\ 0100_2 = 04_{16} \\ 12_{10} & = & 0000\ 1100_2 = 0C_{16} \\ 255_{10} & = & 1111\ 1111_2 = FF_{16} \end{array}$$



HEX Addresses

We won't discuss how a byte recognizes its own address; this is buried deep inside the integrated electronics of the IC chips. The address buss consists of 16 lines, 0 through 15. PET needs this many lines to address all 65,536 bytes. Because location 0 (zero) is included, the maximum address obtainable is 65,535 in decimal. When this location is addressed, all 16 lines of the address buss will have ahigh voltage. In other words logic 1.

	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
65,535 =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

$$= 2^{15} + 2^{14} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^{9} + 2^{8} + 2^{7} + 2^{6} + 2^{5} + 2^{4} + 2^{3} + 2^{2} + 2^{1} + 2^{0}$$

On PET = 2↑15 + 2↑14+... 2↑0 Try it.

....18/

If we now split the 16 columns into four fields of four we can also represent each field using a hexadecimal character thus converting decimal to hexadecimal as Jim has on the left of the memory map.

$65,535_{10}$ = 1111 1111 1111 1111 $_{(2)}$

= $FFFF_{16}$

Recall e.g. #1 of the SYS command (pg.8)

52764_{10} = 1100 1110 0001 1100 $_{(2)}$

= $CE1C_{16}$

When operating PET, the decimal addresses are used for PEEK, POKE and SYS. Therefore you probably won't find yourself converting from decimal to HEX when using BASIC. However you will need to convert from HEX to decimal when you want to SYS to those ROM subroutines.

$CE1C_{16}$ = 1100 1110 0001 1100

$= 2^{15} + 2^{14} + 2^{11} + 2^{10} + 2^{9} + 2^{4} + 2^{3} + 2^{2}$

ON PET = 2↑15 + 2↑14 + 2↑11 + 2↑10 + 2↑9 + 2↑4 + 2↑3 + 2↑2

$= 52764_{10}$

$F422_{16}$ = 1111 0100 0010 0010

$= 2^{15} + 2^{14} + 2^{13} + 2^{12} + 2^{10} + 2^{5} + 2^{1}$

On PET = 2↑15 + 2↑14 + 2↑13 + 2↑12 + 2↑10 + 2↑5 + 2↑1

$= 62498_{10}$

Try verifying some of the other examples using the same conversion method. With a little practice HEX conversions will be as easy as counting to 'F'.

...19/

CAMERA FAIR:

The T. Eaton Company of Canada is holding their 2nd Annual Camera Fair; a camera clinic for photography enthusiasts. Representatives of 10 camera manufacturers will be there to answer questions and Eaton's has invited Commodore to participate with a PET program clinic. Karl Hildon will be there to demonstrate the new big keyboard PET, answer programming questions and give print-outs of your personal computerized biorhythm. Some club programs will be available for copying (limit 1 per customer).

PLACE: EATON CENTRE, Downtown Toronto (Dundas & Yonge)

Show dates are: April 5th - 11:00 a.m. - 3:00 p.m.
April 6th - 5:00 p.m. - 8:00 p.m.
April 7th - 10:00 a.m. - 5:00 p.m.

Hope to see you there!

FOR YOUR INFORMATION:

The subscription form for the 1979 "TRANSACTOR" bulletin will be in next month's issue....Don't miss it!

Karl J.

COMMERCIAL CONFUSION,
or,
"Where'd the penny go?"

Jim Butterfield
Toronto

PET is certainly the greatest business tool since electric pencil sharpeners, and printers and floppy disks will herald an explosion of commercial apprlications.

Basic seems like the ideal language for a small business system - but it has a hidden "gotcha" that will give you problems if you don't know how to hardle it. I call it, "the missing pennies problem", and it's common to almost all Basic implementations.

Crank up your PET and try this: PRINT 2.23 - 2.18 -- it's a simple business calculation and the answer has gotta be a nickel, right? So how come PET says .0499999998?

Think of the mess this could cause if you're printing out neat columns of dollar-and-cent results. Think of the problems if you arrange to print the first two places behind the decimal point: you'll print .04 instead of .05! Think of what the auditor will say when he finds that the totals don't add up correctly!

In a moment we'll discuss how to get rid of this problem. First, though, let's see how it happens.

PET holds numbers in floating binary. That means certain fractions don't work out evenly. Just as, in decimal, one third works out to .333333 .., an endless number, PET sees fractions like .10 or .60 as endless repeating fractions - in binary. So fit the fraction in memory, it must trim it. Thus, many fractions such as .37 are adjusted slightly before storage.

Try this program: it will tell you how numbers are stored inside PET:

```
100 INPUT"AMOUNT";A:B=INT(A):C=A-B:?A;"=";B;" ";
110 FOR J=1TO10:C=C*10:D=INT(C):C=C-D:?D;:IF C>0 THEN NEXT J
120 ?:GOTO100
```

If you try entering numbers in our above example, 2.23 and 2.18, you'll see how PET stores them - and why the problems happen.

How to fix the problem. Easy. Change all numbers to pennies - which eliminates fractions - and your troubles disappear. For example:

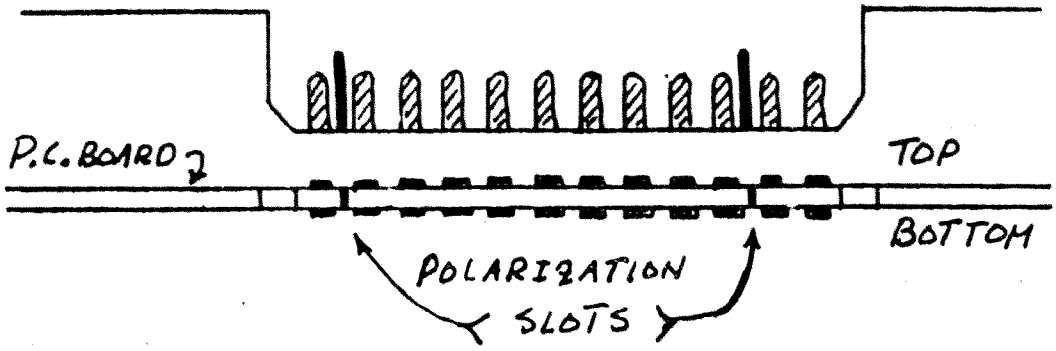
```
340 INPUT"AMOUNT";A : A=INT(A*100+.5)  converts A to pennies;
          .............
760 PRINT A/100       outputs pennies in dollars-and-cents
```

```
100 DIMC(5),D(13)
110 PRINT"ONTARIO INCOME TAX 1978 TAXATION YEAR":PRINT"  J BUTTERFIELD"
111 INPUT"INSTRUCTIONS";Z$:IFASC(Z$)=78GOTO120
112 PRINT"ONTARIO INCOME TAX FOR 1978"
113 PRINT"FOLLOW YOUR FORM: THIS PROGRAM WILL"
114 PRINT"HELP WITH THE ARITHMETIC."
115 PRINT"FOR 'NIL' ITEMS, JUST PRESS 'RETURN'."
116 PRINT"FOR 'MULTIPLE' ENTRIES, ENTER AMOUNT"
117 PRINT"AND PRESS '+' INSTEAD OF 'RETURN' TO"
118 PRINT"SIGNAL MORE ITEMS TO COME."
120 DEFFNS(M)=(M+S-ABS(M-S))/2
130 DEFFNB(M)=(M+B+ABS(M-B))/2
140 DEFFNP(M)=INT(M*P/100+.49)
150 DEFFNI(M)=INT(M*100+.5)
200 P1=1:C=1:F=1:GOSUB2000
210 I$="INCOME FROM EMPLOYMENT":GOSUB2100
220 P=3:S=25E3:I=FNS(FNP(I)):F=-1:I$="LESS EMPLOYMT EXPENS":GOSUB2150
230 F=0:I=C(C):I$="*NET EMPL EARNINGS":GOSUB2200
240 F=1:I$="DIVD FROM CDN CORP":GOSUB2100
250 D(0)=I:I$="INTEREST & INV INCM":GOSUB2100
255 D(1)=I:I$="CANADIAN CAPITAL GAINS":GOSUB2100
260 D(4)=I:I$="ALL OTHER INCOME":GOSUB2100
270 I=C(1):I$="**TOTAL INCOME**":GOSUB2200
280 D(2)=I:P1=2:GOSUB2000:GOSUB2200
282 PRINT"LESS:":C=2:C(C)=0:I$="CPP CONTRIBUTIONS":GOSUB2100
284 I$="UIC PREMIUMS":GOSUB2100
286 I$="OTHER DEDUCTIONS":GOSUB2100
290 I=C(C):C(C)=0:C=1:F=-1:I$=" *TOT DEDUCTIONS:":GOSUB2150
300 I=C(C):I$=" *NET INCOME*":D(3)=I:GOSUB2200
310 PRINT"EXEMPTIONS:":C=2:I$="BASIC EXEMPTION":F=1:I=243E3:GOSUB2150
320 I$="AGE EXEMPT":GOSUB2100:I$="MARRIED EXEMPT":GOSUB2100
330 I$="DEPNDT CHILD EXMPT":GOSUB2100:I$="OTHER EXMPT":GOSUB2100
340 I=C(C):D(12)=I:C=1:F=-1:I$=" *TOTAL EXEMPT*":GOSUB2150
350 GOSUB2300:I$=" **LINE 46**":GOSUB2200
360 C=2:I$="MEDICAL EXPENSES":GOSUB2100
370 C(C)=I:IFI=0GOTO400
380 F=-1:P=3:I=FNP(D(3)):I$="*LESS 3% N.I.":GOSUB2150
390 GOSUB2300:I$="ALLOWABLE MED EXP":GOSUB2200
400 F=1:I$="CHARITABLE DONATNS":GOSUB2100:I=C(C):C(C)=0
410 B=1E4:I=FNB(I):I$="**STANDARD DEDCTN**":GOSUB2150
420 S=1E5:I=FNS(D(0)+D(1)+D(4)):D(10)=I
430 IFI>0THENI$="*I, D & CG DEDUCTION":GOSUB2150
440 I$="ALL OTHER DEDUCTIONS":GOSUB2100
450 I=C(C):C(C)=0:C=1:F=-1:I$="**TOTAL DEDUCTIONS":GOSUB2150
460 GOSUB2300:I$=" **TAXABLE INCOME**":GOSUB2200:D(5)=I:PRINT
470 IFI<231E3THENPRINT" NO TAX PAYABLE":D(5)=0:GOTO1000
480 IFI<=24E5ANDD(0)=0THENPRINT"YOU MAY USE TAX TABLE .. OR.. "
490 DATA36504,9521,36
500 P1=1:GOSUB2010
510 DATA91260,30192,43
520 DATA59319,17735,39
530 DATA36504,9521,36
540 DATA21294,4654,32
550 DATA16731,3377,28
560 DATA13689,2616,25
570 DATA10647,1916,23
580 DATA7605,1278,21
590 DATA4563,700,19
```

```
600 DATA3042,426,18
610 DATA1521,167,17
620 DATA761,46,16
630 DATA-1E10,0,6
640 DATA-1
650 READX,Y,P:IFI<X*102GOTO650
660 T=Y*100:PRINT"ON FIRST $";X;"TAX IS       ";Y
670 J=I-X*100:I=FNP(J):PRINT"ON RMG $";J/100;"TAX AT";P;"% IS $";I/100
680 I=I+T:C=3:F=1:I$="TOTAL FED INCM TAX":GOSUB2150
690 S=I:P=25:I=FNS(FNP(D(0))):D(11)=I
695 IFI>0THENF=-1:I$="DIV TAX CREDIT":GOSUB2150
700 I=C(C):I$=" *BASIC FEDERAL TAX*":GOSUB2200:PRINT:D(6)=I
710 IFI<3E4GOTO740
720 IFI<=3333E2THENI=3E4:GOTO740
730 P=9:I=FNP(I)
740 C=4:C(C)=0:F=1:I$="GENERAL TAX REDUCTION":GOSUB2150
750 I$="REDUCTION FOR CHILDREN":GOSUB2100
760 S=5E4:I=FNS(C(C)):C(C)=0:C=3:F=-1:I$="TOTAL REDUCIONS":GOSUB2150
770 GOSUB2300:I$=" **FEDERAL TAX**":GOSUB2200:D(7)=I:D(8)=I
780 C=4:I$="FOREIGN TAX PAID":GOSUB2100:IFI=0GOTO850
790 W=I:I$="FORGN INCOME":GOSUB2100:K=I:X=(D(3)-D(10))/100:Y=(D(7)+D(11))/100
800 S=INT(K/X*Y):PRINTK/100;"/";X;"*";Y;"=";S/100
810 I$="--DEDUCT:":I=FNS(W):GOSUB2200:D(8)=D(8)-I
820 PRINT"..ANOTHER COUNTRY...":GOTO780
850 I=D(8):I$="FEDERAL TAX PAYABLE":PRINT:GOSUB2200:PRINT
860 P=44:I=FNP(D(6)):I$="BASIC ONTARIO TAX":GOSUB2200:D(9)=I
1000 READX:IFX<>-1GOTO1000
1010 PRINT"==ONTARIO PROPERTY TAX=="
1020 I$="TOTAL RENT PAYMENTS":GOSUB2100:IFI=0GOTO1040
1030 P=20:I=FNP(I):I$="*20% OF RENT":GOSUB2200
1040 C=1:C(C)=I:F=1:I$="PROPERTY TAXES&COLLG RES":GOSUB2100
1050 I=C(C):P=10:X=FNP(I):I$="*OCCUPANCY COST*":GOSUB2200:PRINT
1060 S=18E3:I=FNS(I):I$=" ADD ":GOSUB2200:C(C)=I:I=X:I$=" TO ":GOSUB2150
1070 I$="PROPERTY TAX CREDIT":I=C(C):GOSUB2200
1080 P=1:I=FNP(D(12)):I$="SALES TAX CREDIT":GOSUB2150
1090 I$="PENSIONER CREDIT":GOSUB2100:I=C(C):I$="TOTAL CREDITS":GOSUB2200
1100 P=2:I=FNP(D(5)):I$="LESS--":F=-1:GOSUB2150
1110 GOSUB2300:S=5E4:I=FNS(I):I$="ONTARIO P S & P CREDITS":GOSUB2200
1120 C(C)=I:I$="POLITICAL TAX CREDIT":GOSUB2100
1130 I=C(C):I$="*TOTAL ONT TAX CREDITS":D(13)=I:GOSUB2200
1140 P1=4:GOSUB2000:I=D(8):I$="FEDERAL TAX PAYABLE":GOSUB2200
1150 I$="POLIT/BUS/EMPLMT CREDIT":GOSUB2100:X=D(8)+D(9)-I
1160 I$="ONTARIO TAX PAYABLE":I=D(9):GOSUB2200
1170 I$="TOTAL PAYABLE":I=X:GOSUB2200:PRINT
1180 C=1:C(C)=0:F=1:I$="TAX DEDUCTED PER SLIPS":GOSUB2100
1190 I$="ONTARIO TAX CREDITS":I=D(13):GOSUB2150
1200 I$="OVERPAYMENTS/INSTALMENTS":GOSUB2100
1210 I=C(C):I$="**TOTAL CREDITS**":GOSUB2200:PRINT
1220 I$="BALANCE DUE":I=X-I:IFI<0THENI$="REFUND:":I=ABS(I)
```

```
 1230 GOSUB2200:PRINT
 1999 END
 2000 PRINT:PRINT"===PAGE";:GOTO2020
 2010 PRINT:PRINT"===SCHEDULE";
 2020 PRINTP1;"OF RETURN===":RETURN
 2100 I=0:GETZ$:PRINTI$;"? ";
 2110 Y$="":PRINT"&";
 2120 GETZ$:IFZ$=""GOTO2120
 2130 Z=ASC(Z$):IFZ>47ANDZ<58GOTO2145
 2133 IFZ$="-"ANDY$=""GOTO2145
 2136 IFZ$="."GOTO2145
 2139 IF(Z=157ORZ=20)ANDY$<>""THENY$=LEFT$(Y$,LEN(Y$)-1):PRINT" ";:GOTO2147
 2140 IFZ$="+"THENPRINT"";:I=I+VAL(Y$):FORJ=1TOLEN(Y$):PRINT"";:NEXT:GOTO2110
 2142 IFZ=13ANDI=0THENPRINT"";
 2143 IFZ=13THENI=FNI(I+VAL(Y$)):PRINT:GOTO2150
 2144 GOTO2120
 2145 Y$=Y$+Z$
 2147 PRINTZ$;:GOTO2120
 2150 C(C)=C(C)+I*F
 2200 PRINTI$;:M=1E8:FORJ=LEN(I$)TO25:PRINT" ";:NEXTJ:J=ABS(I):Z$=" ":Z=0
 2210 D=INT(J/M):J=J-D*M:IFD=ZTHENPRINT" ";:GOTO2230
 2220 Z$=",":Z=10:PRINTCHR$(D+48);
 2230 M=M/10:IFM=1E4THENPRINTZ$;
 2240 IFM=10THENPRINT".";:Z=M
 2250 IFM>=1GOTO2210
 2260 IFI<0THENPRINT"CR";
 2270 PRINT:RETURN
 2300 B=0:I=FNB(C(C)):C(C)=I:RETURN
READY.
```

USER PORT COOKBOOK

Recall last month the User Port Cookbook was included with the "Transactor". Since then an error has been brought to my attention. The cardedge illustration of the User Port on the front page is reversed. Compare it to the actual cardedge at the back of your PET and notice the polarization slots. Although the pin names correspond correctly, the illustration is essentially the "mirror image". Below is the corrected version which you can cut out and paste over top.

NAME	GND	CA1	PA0	PA1	PA2	PA3	PA4	PA5	PA6	PA7	CB2	GND
PIN	A	B	C	D	E	F	H	J	K	L	M	N